```
            ASSERT( 0 );
            bRc = FALSE;
            break;
         }
      }

      ////////////////////////////////////////////////////////////////
      // Now save site page include-exclude list in the placement_sites table
      ////////////////////////////////////////////////////////////////
      pos = targetPages.GetStartPosition();
      DWORD dwPageID;
      while (pos)
      {
         targetPages.GetNextAssoc( pos, dwPageID, bJunk );

         wsprintf( buf, "insert placement_pages(ad_id,page_id,include) values(%d,%d,%d)",
                   adID, dwPageID, includePages );

         if (iafmain.exec( buf ) != 1)
         {
            ASSERT( 0 );
            bRc = FALSE;
            break;
         }
      }

      break;
   }

   iafmain.commit();
   return( bRc );
}

BOOL Ad::Remove( BOOL bRemoveFromPlacements )
{
   char buf[1024];
   BOOL bRc = TRUE;

   while (TRUE)
   {
      ////////////////////////////////////////////////////////////////
      // Delete locations from the "placement_locations" table
      ////////////////////////////////////////////////////////////////
      wsprintf( buf, "delete placement_locations where ad_id=%d", id );
      if (iafmain.execErrOK( buf ) != 0)
      {
         ASSERT( 0 );
         bRc = FALSE;
         break;
      }

      ////////////////////////////////////////////////////////////////
      // Delete the SICs from the "placement_sics" table
      ////////////////////////////////////////////////////////////////
      wsprintf( buf, "delete placement_sics where ad_id=%d", id );
      if (iafmain.execErrOK( buf ) != 0)
      {
         ASSERT( 0 );
         bRc = FALSE;
         break;
      }

      ////////////////////////////////////////////////////////////////
      // Delete the site categories from the placement_sitecats table
      ////////////////////////////////////////////////////////////////
      wsprintf( buf, "delete placement_sitecats where ad_id=%d", id );
      if (iafmain.execErrOK( buf ) != 0)
      {
         ASSERT( 0 );
         bRc = FALSE;
         break;
      }

      ////////////////////////////////////////////////////////////////
      // Delete the user interests from the placement_interests table
```



```
      ////////////////////////////////////////////////////////////////
      wsprintf( buf, "delete placement_interests where ad_id=%d", id );
      if (iafmain.execErrOK( buf ) != 0)
      {
         ASSERT( 0 );
         bRc = FALSE;
         break;
      }

      ////////////////////////////////////////////////////////////////
      // Delete the site include-exclude list from the placement_sites table
      ////////////////////////////////////////////////////////////////
      wsprintf( buf, "delete placement_sites where ad_id=%d", id );
      if (iafmain.execErrOK( buf ) != 0)
      {
         ASSERT( 0 );
         bRc = FALSE;
         break;
      }

      ////////////////////////////////////////////////////////////////
      // Delete the site page include-exclude list from the placement_sites table
      ////////////////////////////////////////////////////////////////
      wsprintf( buf, "delete placement_pages where ad_id=%d", id );
      if (iafmain.execErrOK( buf ) != 0)
      {
         ASSERT( 0 );
         bRc = FALSE;
         break;
      }

      if (bRemoveFromPlacements)
      {
         ////////////////////////////////////////////////////////////////
         // Lastly, delete the placement from the placements table
         ////////////////////////////////////////////////////////////////
         wsprintf( buf, "delete placements where id = %d", id );
         if (iafmain.execErrOK( buf ) != 0)
         {
            ASSERT( 0 );
            bRc = FALSE;
            break;
         }
      }
      break;
   }

   iafmain.commit();
   return( bRc );
}

void Ad::Reset()
{
   dayeOfWeek = 0x7f;
   flags = Production | SpreadEvenly;
   frequency = 0;
   imageSeries = FALSE;
   maxImpressions = 0;
   type = Normal;
   domainType = 0;
   gender = 0;
   maxAmount = 0;
   szPONumber.Empty();
   startTime = 0;
   endTime = 0;
   os = DefaultMask;
   browser = DefaultMask;
   domainType = DefaultMask;
   isp = DefaultMask;
   hoursOfDay = 0xffffff;
   nEmployees = DefaultMask;
   salesVolume = DefaultMask;
   gender = DefaultMask;
   includePages = 0;
   includeSites = 0;
```

```
    seriesNext = 0;

    delete [] sicCodes;
    nSICCodes = 0;
    sicCodes = NULL;

    delete [] locations;
    nLocations = 0;
    locations = NULL;

    targetPages.RemoveAll();
    targetSites.RemoveAll();
    siteCategories.RemoveAll();
    interests.RemoveAll();

    adDescription.Empty();
    fileName.Empty();
    jumpTo.Empty();
  }
}
```

```
BOOL Ad::Book( DWORD advertiserID )
{
   char buf[1024];
   char szTime[ 10 ];

   if (!advertiserID)
   {
      ASSERT( 0 );
      return( FALSE );
   }

   //
   // If this is a barter ad, set max_impressions = to 1
   if (type == Barter)
   {
       maxImpressions = 1;
   }

   strcpy( buf, "insert placements(jumpto,max_impressions,type,os,browser,domainType,isp,frequ
            "image_series,advertiser,flags,hours_of_day,days_of_week,employees,sales,descri
            "max_amount,po_number,gender,active,approved,filename" );
   if (startTime)
      strcat( buf, ",start_time" );

   if (endTime)
      strcat( buf, ",end_time" );

   strcat( buf, ") values(" );

   addValue( buf, jumpTo );
   addValue( buf, maxImpressions );
   addValue( buf, type );
   addValue( buf, os );
   addValue( buf, browser );
   addValue( buf, domainType );
   addValue( buf, isp );
   addValue( buf, frequency );
   addValue( buf, imageSeries );
   addValue( buf, advertiserID );
   addValue( buf, flags );
   addValue( buf, hoursOfDay );
   addValue( buf, daysOfWeek );
   addValue( buf, nEmployees );
   addValue( buf, salesVolume );
   addValue( buf, adDescription );
   addValue( buf, maxAmount );
   addValue( buf, szPONumber );
   addValue( buf, gender );
   addValue( buf, active );
   addValue( buf, approved );
   addValue( buf, fileName, FALSE );

   if (startTime)
   {
      strftime( szTime, 9, "%m/%d/%y", gmtime( &startTime ) );
      strcat( buf, "," );
      addValue( buf, szTime, FALSE );
   }

   if (endTime)
   {
      strftime( szTime, 9, "%m/%d/%y", gmtime( &endTime ) );
      strcat( buf, "," );
      addValue( buf, szTime, FALSE );
   }

   strcat( buf, ")" );

   if (!isfmain.exec( buf ) != 1)
   {
      ASSERT( 0 );
      return( FALSE );
   }

   //
```



```
   // Get the ID of the newly added ad
   int   adID = 0;
   {
      Cursor   c;
      c.bind( SQL_C_LONG, &adID, 4 );
      strcpy( buf, "select max(id) from placements" );
      c.exec( buf );
      c.fetchNext();
      isfmain.commit();
   }

   if (!adID)
   {
      ASSERT( 0 );
      return( FALSE );
   }

   return( AddPlacementTables( adID ) );
}

BOOL Ad::Update()
{
   //
   // To update an ad, we delete the existing ad
   // and re-book it.
   if (Remove( FALSE ))
   {
      //
      // Re-determine if the ad is targeted
      double dPerAdCost = CalculateCostPerAd();
      if (dPerAdCost == BASE_AD_COST)
      {
         flags &= Ad::Targeted;
      }
      else
      {
         flags |= Ad::Targeted;
      }

      char buf[1024];
      char szTime[ 10 ];

      strcpy( buf, "update placements set " );

      //
      // Don't update max_impressions if this is a barter ad.  REP.EXE
      // credits the placement so we don't want to overwrite the
      // barter credits
      if (type != Barter)
      {
         strcat( buf, "max_impressions=" );      addValue( buf, maxImpressions );
      }
      strcat( buf, "jumpto=" );            addValue( buf, jumpTo );
      strcat( buf, "type=" );                    addValue( buf, type );
      strcat( buf, "os=" );                      addValue( buf, os );
      strcat( buf, "browser=" );                 addValue( buf, browser );
      strcat( buf, "domainType=" );              addValue( buf, domainType );
      strcat( buf, "isp=" );                     addValue( buf, isp );
      strcat( buf, "frequency=" );               addValue( buf, frequency );
      strcat( buf, "image_series=" );            addValue( buf, imageSeries );
      strcat( buf, "flags=" );                   addValue( buf, flags );
      strcat( buf, "hours_of_day=" );            addValue( buf, hoursOfDay );
      strcat( buf, "days_of_week=" );            addValue( buf, daysOfWeek );
      strcat( buf, "employees=" );               addValue( buf, nEmployees );
      strcat( buf, "sales=" );                   addValue( buf, salesVolume );
      strcat( buf, "description=" );             addValue( buf, adDescription );
      strcat( buf, "max_amount=" );              addValue( buf, maxAmount );
      strcat( buf, "po_number=" );               addValue( buf, szPONumber );
      strcat( buf, "gender=" );                  addValue( buf, gender );
      strcat( buf, "active=" );                  addValue( buf, active );
      strcat( buf, "approved=" );              addValue( buf, approved );
      strcat( buf, "filename=" );              addValue( buf, fileName );

      strcat( buf, "start_time=");
      if (startTime)
```

```
      {
         strftime( szTime, 9, "%m/%d/%y", gmtime( &startTime ) );
         addValue( buf, szTime );
      }
      else
      {
         strcat( buf, "(null)," );
      }

      strcat( buf, "end_time=");
      if (endTime)
      {
         strftime( szTime, 9, "%m/%d/%y", gmtime( &endTime ) );
         addValue( buf, szTime, FALSE );
      }
      else
      {
         strcat( buf, "(null)" );
      }

      strcat( buf, "where id=" );
      addValue( buf, id, FALSE );

      if (iafmain.exec( buf ) != 1)
      {
         ASSERT( 0 );
         return( FALSE );
      }

      return( AddPlacementTables( id ) );
   }

   return( FALSE );
}

BOOL Ad::AddPlacementTables( DWORD adID )
{
   char buf[1024];
   BOOL bRc = TRUE;

   while (TRUE)
   {
      ////////////////////////////////////////////////////////////////
      // Now save the locations to the "placement_locations" table
      ////////////////////////////////////////////////////////////////
      for (int nLoop = 0; nLoop < nLocations; nLoop++)
      {
         strcpy( buf, "insert placement_locations(" );

         if (locations[nLoop].country)
            strcat( buf, "country," );
         if (!locations[nLoop].state.IsEmpty())
            strcat( buf, "state," );
         if (!locations[nLoop].zipCode.IsEmpty())
            strcat( buf, "zipcode," );
         if (locations[nLoop].areaCode)
            strcat( buf, "areacode," );

         strcat( buf, "ad_id) values(" );

         if (locations[nLoop].country)
            addValue( buf, locations[nLoop].country );
         if (!locations[nLoop].state.IsEmpty())
            addValue( buf, locations[nLoop].state );
         if (!locations[nLoop].zipCode.IsEmpty())
            addValue( buf, locations[nLoop].zipCode );
         if (locations[nLoop].areaCode)
            addValue( buf, locations[nLoop].areaCode );

         addValue( buf, adID, FALSE );
         strcat( buf, ")" );

         if (iafmain.exec( buf ) != 1)
         {
            ASSERT( 0 );
```



```
            bRc = FALSE;
            break;
         }
      }

      ////////////////////////////////////////////////////////////////
      // Now save the SICs to the "placement_sics" table
      ////////////////////////////////////////////////////////////////
      for (nLoop = 0; nLoop < nSICCodes; nLoop++)
      {
         wsprintf( buf, "insert placement_sics(ad_id,siccode) values(%d,'%s')",
                   adID, sicCodes[ nLoop ].asText() );

         if (iafmain.exec( buf ) != 1)
         {
            ASSERT( 0 );
            bRc = FALSE;
            break;
         }
      }

      ////////////////////////////////////////////////////////////////
      // Now save the site categories to the placement_sitecats table
      ////////////////////////////////////////////////////////////////
      POSITION pos = siteCategories.GetStartPosition();
      DWORD    dwInterestID;
      BOOL     bJunk;
      while (pos)
      {
         siteCategories.GetNextAssoc( pos, dwInterestID, bJunk );

         wsprintf( buf, "insert placement_sitecats(ad_id,interest_id) values(%d,%d)",
                   adID, dwInterestID );

         if (iafmain.exec( buf ) != 1)
         {
            ASSERT( 0 );
            bRc = FALSE;
            break;
         }
      }

      ////////////////////////////////////////////////////////////////
      // Now save the user interests to the placement_interests table
      ////////////////////////////////////////////////////////////////
      pos = interests.GetStartPosition();
      while (pos)
      {
         interests.GetNextAssoc( pos, dwInterestID, bJunk );

         wsprintf( buf, "insert placement_interests(ad_id,interest_id) values(%d,%d)",
                   adID, dwInterestID );

         if (iafmain.exec( buf ) != 1)
         {
            ASSERT( 0 );
            bRc = FALSE;
            break;
         }
      }

      ////////////////////////////////////////////////////////////////
      // Now save site include-exclude list in the placement_sites table
      ////////////////////////////////////////////////////////////////
      pos = targetSites.GetStartPosition();
      DWORD dwSiteID;
      while (pos)
      {
         targetSites.GetNextAssoc( pos, dwSiteID, bJunk );

         wsprintf( buf, "insert placement_sites(ad_id,site_id,include) values(%d,%d,%d)",
                   adID, dwSiteID, includeSites );

         if (iafmain.exec( buf ) != 1)
         {
```

```
// sitepage.cpp
//

#include "stdafxh"
#include "objects.h"
#include "/d/toolkit/db.h"
#include "/d/toolkit/iaf_util.h"
#include "/d/toolkit/dbutil.h"

void message(const char *s);

SitePage::SitePage()
{
   id = 0;
   siteID = 0;
    categorized = FALSE;
}

void SitePage::loadCategories()
{
   DWORD interestID;
   Cursor c;
   c.bind(SQL_C_LONG, &interestID, sizeof(interestID));
   char sql[1024] = "select interests_id from page_categories where page_id=";
   addValue(sql, id, FALSE);
   strcat(sql, " union all select interests_id from site_categories where site_id=");
   addValue(sql, siteID, FALSE);
   c.exec(sql);
   while( c.fetchNext() ) {
      categories.Add(interestID);
   }
}

extern BOOL defaultAdsMode;

SitePage* SitePage::lookupPage(Database& db, const char *from, const char *requestHdr)
{
   // from key format: sitekey/docname
   if( from == 0 )
      return 0;

   if( strnicmp(from, "www.", 4) == 0 )
      from += 4;

   if( *from == 0 )
      return 0;
   const char *q = strchr(from, '/');
   if( q == 0 || strlen(from) > 75 )
      return 0;

   CString key;
   {
      // truncate a unique number from the end of the key
      const char *lastSlash = strrchr(q - 1, '/');
      if( lastSlash && isdigit(lastSlash[1]) )
         key = CString(from, lastSlash - from);
      else
         key = from;
      if( key.GetLength() > 64 )
         key = key.Left(64);  // truncate to column width
   }

   SitePage *p = new SitePage;

   {
      Cursor c(db);
      c.bind(SQL_C_LONG, &p->id, 4);
      c.bind(SQL_C_LONG, &p->siteID, 4);
      c.bind(SQL_C_LONG, &p->categorized, 4);
      char sql[1024] =
         "select id,site,categorized from sitepages where keyname=";
      addValue(sql, key, FALSE);
      c.exec(sql);
      if( c.fetchNext() ) {
         return p;
      }
```



```
      }
   }

   // Didn't find the page.  Add page if site is correct.
   {
      CString siteKey(from, q - from);
      int approved = 0;
      Cursor c(db);
      c.bind(SQL_C_LONG, &p->siteID, sizeof(p->siteID));
      c.bind(SQL_C_LONG, &approved, sizeof(approved));
      CString sql = "select id,approved from sites where keyname='";
      sql += siteKey + '\'';
      c.exec(sql);
      if( c.fetchNext() ) {
         if( approved == 0 ) {
            message( CString("unapproved site: ") + from );
         }
         else {
            p->add(db, key);
         }
      }
      else {
         delete p;
         p = 0;
         if( !defaultAdsMode )
            message( CString("unknown site: ") + from );
      }
   }

   return p;
}

void SitePage::add(Database& db, const char *keyname)
{
   char buf[512] = "insert sitepages(junk, keyname, site, categorized) values('',";
   addValue(buf, keyname);
   addValue(buf, (int) siteID);
   addValue(buf, (int) categorized, FALSE);
   strcat(buf, ")");
   if( db.exec(buf) != 1 ) {
      TRACE("error adding sitekey\n");
      CString s = "sql: ";
      s += buf;
      ASSERT(FALSE);
      TRACE(s);
      message(s);
   }

   Cursor c(db);
   id = 0;
   c.bind(SQL_C_LONG, &id, 4);
   strcpy(buf, "select id from sitepages where keyname=");
   addValue(buf, keyname, FALSE);
   c.exec(buf);
   if( !c.fetchNext() ) {
      ;
   }
}
```

```
// ad.cpp
//

#include "stdafx.h"
#include <strstrea.h>
#include <fstream.h>
#include "winsock.h"
#include "objects.h"
#include "/d/toolkit/iaf_util.h"
#include "/d/toolkit/db.h"
#include "/d/toolkit/dbutil.h"
#include "/d/derive/sqlderive.h"
#include "/d/newsderive/sig.h"
#include "rememberad.h"

const CString gifsRootDir = "c:\\ian\\ads\\";

#if !defined(_DERIVE)
int nAds() { return ads.GetSize(); }
#endif

extern Database iafmain;

//-----------------------------------------------------------------------
// Ad

Ad::~Ad()
{
   delete[] locations;
   delete[] sicCodes;
}

Ad::Ad(const Ad& ad) :
   started(ad.started),
   id(ad.id), fileName(ad.fileName), jumpTo(ad.jumpTo),
   type(ad.type), os(ad.os), browser(ad.browser),
   domainType(ad.domainType), isp(ad.isp),
   maxImpressions(ad.maxImpressions), nShown(ad.nShown),
   nLocations(ad.nLocations), nSICCodes(ad.nSICCodes),
   frequency(ad.frequency), imageSeries(ad.imageSeries),
   seriesNext(ad.seriesNext), startTime(ad.startTime), endTime(ad.endTime),
   si(ad.si), flags(ad.flags),
   hoursOfDay(ad.hoursOfDay), daysOfWeek(ad.daysOfWeek),
   nEmployees(ad.nEmployees), salesVolume(ad.salesVolume),
   gender(ad.gender), adDescription(ad.adDescription),
   maxAmount(ad.maxAmount), szPONumber(ad.szPONumber),
   active(ad.active), includeSites(ad.includeSites),
   includePages(ad.includePages), approved(ad.approved),
   nJumps(ad.nJumps)
{
   stripSpaces(fileName);
   stripSpaces(jumpTo);

   locations = 0;
   if( nLocations ) {
      locations = new Region[nLocations];
      for( int i = 0; i < nLocations; i++ ) {
         locations[i] = ad.locations[i];
      }
   }

   sicCodes = 0;
   if( nSICCodes ) {
      sicCodes = new SICCode[nSICCodes];
      for( int i = 0; i < nSICCodes; i++ ) {
         sicCodes[i] = ad.sicCodes[i];
      }
   }
}

void Ad::calcSI()
{
   if( maxImpressions == 0 )
      return;
```



```
   time_t t;
   DWORD totalSpan = endTime - startTime;
   if( totalSpan == 0 )
      totalSpan = 1;
   DWORD span = time(&t) - startTime; if( span == 0 ) span = 1;

   si =
   (DWORD) ((double) nShown /
      ((double) span / totalSpan) /
      maxImpressions) * 1000);
}

void Ad::shown()
{
   nShown++;

// if( nShown % 8 == 0 ) {
//    // update SI
//    calcSI();
// }
}

Ad::Ad()
{
   daysOfWeek = 0x7f;
   started = FALSE;
   flags = Production | SpreadEvenly;
   si = 1100;
   sicCodes = 0;
   nSICCodes = 0;
   frequency = 0;
   imageSeries = FALSE;
   id = 0;
   maxImpressions = 0;
   nShown = 0;
   nJumps = 0;
   type = Normal;
   nLocations = 0;
   locations = 0;
   gender = 0;
   maxAmount = 0;
   active = 0;
   approved = 0;
   includePages = 0;
   includeSites = 0;
   startTime = 0;
   endTime = 0;
   os = DefaultMask;
   browser = DefaultMask;
   domainType = DefaultMask;
   isp = DefaultMask;
   hoursOfDay = 0xffffff;
   nEmployees = DefaultMask;
   salesVolume = DefaultMask;
   gender = DefaultMask;
   seriesNext = 0;
}

CString Ad::getFileName()
{
   if( !imageSeries || seriesNext <= 1 )
      return fileName;

   char buf[256];
   wsprintf(buf, "%sld.gif", (const char *) fileName.Left( fileName.GetLength() - 4), seriesNe
   return buf;
}

CString Ad::fullName()
{
   return gifsRootDir + getFileName();
}

#if !defined(_ADSVP)
```

```
// users.cpp
//

#include "stdafx.h"
#include "objects.h"
#include "/d/toolkit/db.h"
#include "/d/toolkit/iaf_util.h"
#include "/d/toolkit/dbutil.h"

/* implementation for hash tables
User* User::_lookupUserByID(DWORD userID)
{
   User *u = new User;
   return u;
}

User* User::_lookupUserByAddress(DWORD ip)
{
   DWORD userID = networkNodeTable.getUserID(ip, FALSE);
   if( userID == 0 ) {
      // Try to get domain info at least.  Note: if user is uniquely
      // identifiable, derive data process will create a record for the
      // user as soon as it gets a chance.
      userID = networkNodeTable.getUserID(justNetworkNumber(ip), TRUE);
   }
   if( userID ) {
      return _lookupUserByID(userID);
   }
   return 0;
}
*/

class UserCursor : public Cursor
{
public:
   UserCursor(Database& db, User *_u) : Cursor(db),
      u(_u) { }

   // just gets field that aren't derivable from request header
   void minimalBind()
   {
      bind( SQL_C_LONG, &u->ftpTried, sizeof(BOOL) );
      bind( SQL_C_LONG, &u->hasCookie, sizeof(BOOL) );
   }

   User  *u;
};

void User::lookupAncillaryInfo(Database& db)
{
   if( userID == 0 ) {
      return;
   }

   Cursor c(db);
   char sql[128];
   wsprintf(sql, "select email from users where id=%ld", userID);
   c.bind(emailAddr);
   c.exec(sql);
   c.fetchNext();
   db.commit();
}

User* User::_lookupUserByID(Database& db, DWORD userID, BOOL *timedOut)
{
   User *u = new User;
   UserCursor c(db, u);
   c.minimalBind();
   char sql[128];

   wsprintf(sql, "select ftp_tried,has_cookie from users where id=%ld", userID);
   if( timedOut != 0 )
      c.setTimeOut(1);
   c.exec(sql);
```



```
   if( c.timedOut() ) {
      *timedOut = TRUE;
      delete u; u = 0;
   }
   else if( c.fetchNext() ) {
      u->userID = userID;
   } else {
      delete u;
      u = 0;
   }

   return u;
}

User* User::_lookupUserByAddress(Database& db, DWORD ip, BOOL *timedOut)
{
   User *u = new User;
   UserCursor c(db, u);
   c.minimalBind();
   c.bind( SQL_C_LONG, &u->userID, 4 );
   char sql[128];
   wsprintf( sql, "select ftp_tried,has_cookie,id from users where ip='%s'",
             (const char *) sqlIPStr(ip) );
   if( timedOut != 0 )
      c.setTimeOut(1);
   c.exec(sql);

   if( c.timedOut() ) {
      *timedOut = TRUE;
      delete u;
      u = 0;
   }
   else if( !c.fetchNext() ) {
      delete u;
      u = 0;
   }

   return u;
}

void User::updateFTPTried(Database& db)
{
   if( tempUserObject() ) {
      ASSERT(FALSE);
      return;
   }

   char buf[256];
   wsprintf(buf, "update users set ftp_tried=%d where id=%ld",
            ftpTried ? 1 : 0, userID
            );
   db.exec(buf);
   db.commit();
}

void User::makePermanent(Database& db)
{
   if( !tempUserObject() )
      return;

   ASSERT( name.IsEmpty() && title.IsEmpty() && emailAddr.IsEmpty() );

   // add to DB
   char buf[4096] =
      "insert users (ip,browser,bver1,bver2,os,domain_type,is_proxy,is_networkdesc,ftp_tried,h
   strcat(buf, " values (");
   addDinValue(buf, ip);
   addValue(buf, browser);
   addValue(buf, bVer1);
   addValue(buf, bVer2);
   addValue(buf, os);
   addValue(buf, domainType);
   addBool(buf, proxy);
   addBool(buf, isNetworkDescription);
   addBool(buf, ftpTried);
```

```
    addBool(buf, hasCookie, FALSE);
    strcat(buf, ")");

    if( db.doInsert(buf) == 1 ) {
       Cursor c(db);
       c.bind(SQL_C_LONG, &userID, 4);

       strcpy(buf, "select max(id) from users where ip=");
       addBinValue(buf, ip, FALSE);
       c.exec(buf);

       c.fetchNext();

       ASSERT( userID != 0 );
    }

    db.commit();
}
```

```
// server.cpp
//

#include "stdafx.h"
#include <fstream.h>
#include "server.h"
#include "/d/toolkit/sock.h"
#include "/d/toolkit/mapstate.h"
#include "/d/toolkit/tsutil.h"
#if defined(_ADSVR)
#include "getrequest.h"
#include "tables.h"
 void message(const char *);
#elif defined(_IAF)
#include "request.h"
#include "iafrequest.h"
void qrPurge();
void message(const char *) { }
#else
#include "request.h"
#include "mgmtrequest.h"
void message(const char *) { }
#endif

#include "/d/toolkit/crit.h"
extern CriticalSection cast;

const char cHTTPVer[] = "HTTP/1.0 ";
const char cContentLen[] = "Content-Length: ";
const char cErrHeader[] = "<h1>Error ";
const char cErrTrailer[] = "</h1>";
const char cContentHTML[] = "Content-Type: text/html\r\n";

extern int nListenerThreads;

ofstream errLog;

void sendError(Connection *c, const char *msg, const char *headerField)
{
   char buf[10];

   CString s = cHTTPVer;
   s += msg;
   s += "\r\n";
   s += cContentHTML;
   if( headerField )
      s += headerField;
   s += cContentLen;
   int len = strlen(msg) + strlen(cErrHeader) + strlen(cErrTrailer);
   s += itoa(len, buf, 10);
   s += "\r\n\r\n";
   s += cErrHeader;
   s += msg;
   s += cErrTrailer;

   c->write( (const char *) s, s.GetLength() );
}

BOOL addressOK(const sockaddr_in& from)
{
   if( from.sin_addr.S_un.S_un_b.s_b1 == 206 &&
       from.sin_addr.S_un.S_un_b.s_b2 == 4 &&
       from.sin_addr.S_un.S_un_b.s_b3 == 219 )
   {
      // IAP network
      return TRUE;
   }

   return FALSE;
}

void servicePequest(Connection *c, const sockaddr_in& from)
{
// if( !addressOK(from) )
//    return;
```



```
   const BUFLEN = 32768;
   char buf[BUFLEN];
   buf[1] = 0;
   int n = 0;                              // total n bytes read
   const char *p = buf;
   int countDown = 0;                  // Content-length
   Connection::ReadError err = Connection::OK;
   while( 1 ) {
      int toRead = BUFLEN - n - 1;
      int nRead = c->read(buf + n, toRead, err);
      n += nRead;
      buf[n] = 0;
      if( countDown ) {
         countDown -= nRead;
         if( countDown <= 0 )
            break;
      }
      if( nRead == 0 ) {
         // error
         break;
      }
      const char *p;
      if( (p = strstr(buf, "\r\n\r\n")) != 0 ) {
         const char *cl = strstr(buf, "Content-length:");

         if (!cl)
            cl = strstr(buf, cContentLen);

         if( cl ) {
            cl += 15;
            sscanf(cl, "%ld", &countDown);
            countDown -= strlen(p + 4);    // decrement by what we've already got
//          countDown -= n - ((p + 4) - buf); // decrement by what we've already got
            if( countDown > 0 )
               continue;

         }
         break;
      }
   }

   Verb v = UNKNOWN;
   const char *r = buf;
   if( strnicmp(buf, "get ", 4) == 0 ) {
      v = GET;
      r += 4;
   } else if( strnicmp(buf, "head ", 5) == 0 ) {
      v = HEAD;
      r += 5;
   } else if( strnicmp(buf, "post ", 5) == 0 ) {
      v = POST;
      r += 5;
   }

   if( v == UNKNOWN ) {
      if( *buf == 0 ) {
         sendError(c, "400 Bad Request");
         if( buf[1] != 0 ) {
            message("empty request, buf!");
         }
         else if( err == Connection::TimeOut ) {
            message("empty request, timeout");
         }
         else if( err == Connection::PeadErr ) {
            message("empty request, readerr");
         }
         else {
            message("empty request, err=OK?");
         }
      }
      else {
         sendError(c, "501 Not Implemented");
      }
      return;
   }
```

```
#if defined(_IAF)
   IAFRequest gr(c, v, r, from);
#elif defined(_ADSVR)
   GetRequest gr(c, v, r, from);
#else
   MgmtRequest gr(c, v, r, from);
#endif
   gr.service();
}

Listener *listener = 0;

int nThread = 0;
int maxThreads = 1;

UINT listenerThread(LPVOID)
{
   static DWORD sd = GetTickCount();
   srand( sd++ );

   while( 1 ) {
      sockaddr_in from;
      Connection *c = listener->waitForConnection(from);
      if( c ) {
         {
            Crit c(fast);
            int n = ++nThread;
            if( n > maxThreads )
               maxThreads = n;
         }
         serviceRequest(c, from);
         delete c;
         {
            Crit c(fast);
            nThread--;
#if defined(_IAF)
            if( nThread == 0 ) {
               // idle
               qrPurge();
            }
#endif
         }
      }
   }

   return 0;
}

BOOL startServer()
{
#if defined(_ADSVR)
   if( !openTables() ) {
      AfaMessageBox("Error opening tables");
      return FALSE;
   }

   if( !initWinsock() ) {
      return FALSE;
   }

   mapStateInit();
   initCountryTimeZoneTable();
#endif

#if 0
   {
      // TEMP!
      Connection c;
      if( c.connect("www.microsoft.com", 80) ) {
         c.write("GET /sdd HTTP/1.0\r\n\r\n", 22);

         while( 1 ) {
            char buf[256];
            int n = c.read(buf, 255);
```



```
            if( n ) {
               buf[n] = 0;
               TRACE("%s", buf);
            }
            else
               break;
         }
      }
      return TRUE;
   }
#endif

#if defined(_PORT)
   int   port = _PORT;
#else
   int   port = 80;
#endif
   listener = new Listener(port);
   if( listener->ok() ) {

#if defined(_ADSVR)
      errLog.open("c:/ian/errlog.txt",
                  ios::out | ios::app,
                  filebuf::sh_read);
      ASSERT( errLog.is_open() );
      errLog << "------ Ad server started\n"; errLog.flush();
#endif

      for( int i = 0; i <nListenerThreads; i++ ) {
         Sleep(100); // [dam] this is a test; sometimes it doesn't listen right, just a hunch
         AfaBeginThread( listenerThread, 0 );
      }
   }
   else
      ASSERT(FALSE);

   return TRUE;
}
```

```
      Ad *ad = new Ad(rs.ad);
      ad->calcSI();
      if( ad->id == cBadKeyAdID && forTargeting ) {
         delete badKeyErrorAd;
         badKeyErrorAd = ad;
      }
      else {
         ads.Add(ad);
         if( defaultAd == 0 && ad->type != Ad::Test && !ad->isTargeted() )
            defaultAd = ad;
      }
   }

   lafmain.commit();

   // load sites to include/exclude
   for( int i = 0; i < ads.GetSize(); i++ ) {
      Ad& ad = *ads.GetAt(i);
      if( !ad.isTargeted() )
         continue;
      DWORD siteID;
      BOOL include;
      Cursor c;
      c.bind( SQL_C_LONG, &siteID, sizeof(siteID) );
      c.bind( SQL_C_LONG, &include, sizeof(include) );
      char sql[512] = "select site_id,include from placement_sites where ad_id=";
      addValue(sql, ad.id, FALSE);
      c.exec(sql);
      int n = 0;
      while( c.fetchNext() ) {
         if( ad.targetSites.isEmpty() ) {
            ad.targetSites.initHashTable(37);
            ad.includeSites = include;
         }
         ad.targetSites.SetAt(siteID, TRUE);
         n++;
      }
      if( n > 31 ) {
         message("Increase Ad::targetSites hash size");
      }
   }

   if( forTargeting ) {
      // Load site exclusions of placements.  If exclude this ad,
      // and Ad::includeSites is TRUE, remove site from map.  If
      // exclude this ad, and Ad::includeSites is FALSE, add this
      // site to the map.
      for( int i = 0; i < ads.GetSize(); i++ ) {
         Ad& ad = *ads.GetAt(i);
         DWORD siteID;
         Cursor c;
         c.bind(SQL_C_LONG, &siteID, sizeof(siteID));
         char sql[512] = "select site_id from placement_banned where ad_id=";
         addValue(sql, ad.id, FALSE);
         c.exec(sql);
         while( c.fetchNext() ) {
            if( ad.targetSites.isEmpty() ) {
               ad.targetSites.initHashTable(37);
               ad.includeSites = FALSE;   // exclude
            }
            if( ad.includeSites ) {
               ad.targetSites.RemoveKey(siteID);
               if( ad.targetSites.GetCount() == 0 ) {
                  // since map is empty, will go to all sites,
                  //   which is wrong.  Deactivate.
                  ad.startTime = ad.endTime = 0;
                  message( CString("error, no sites allowed for ") + ad.fileName );
               }
            } else
               ad.targetSites.SetAt(siteID, TRUE);
         }
      }
   }
```



```
   // load pages to include/exclude
   for( i = 0; i < ads.GetSize(); i++ ) {
      Ad& ad = *ads.GetAt(i);
      if( !ad.isTargeted() )
         continue;
      DWORD pageID;
      BOOL include;
      Cursor c;
      c.bind( SQL_C_LONG, &pageID, sizeof(pageID) );
      c.bind( SQL_C_LONG, &include, sizeof(include) );
      char sql[512] = "select page_id,include from placement_pages where ad_id=";
      addValue(sql, ad.id, FALSE);
      c.exec(sql);
      int n = 0;
      while( c.fetchNext() ) {
         if( ad.targetPages.isEmpty() ) {
            ad.targetPages.initHashTable(37);
            ad.includePages = include;
         }
         ad.targetPages.SetAt(pageID, TRUE);
         n++;
      }
      if( n > 31 ) {
         message("Increase Ad::targetPages hash size");
      }
   }

   // load site/page categories
   for( i = 0; i < ads.GetSize(); i++ ) {
      Ad& ad = *ads.GetAt(i);
      if( !ad.isTargeted() )
         continue;
      DWORD interestID;
      Cursor c;
      c.bind( SQL_C_LONG, &interestID, sizeof(interestID) );
      char sql[512] = "select interest_id from placement_sitecats where ad_id=";
      addValue(sql, ad.id, FALSE);
      c.exec(sql);
      int n = 0;
      while( c.fetchNext() ) {
         if( ad.siteCategories.isEmpty() ) {
            ad.siteCategories.initHashTable(37);
         }
         ad.siteCategories.SetAt(interestID, TRUE);
         n++;
      }
      if( n > 31 ) {
         message("Increase Ad::siteCategories hash size");
      }
   }

   // load sics
   for( i = 0; i < ads.GetSize(); i++ ) {
      Ad& ad = *ads.GetAt(i);
      if( !ad.isTargeted() )
         continue;

      int n = 0;
      {
         Cursor c;
         c.bind(SQL_C_LONG, &n, sizeof(n));
         char sql[512] = "select count(*) from placement_sics where ad_id=";
         addValue(sql, ad.id, FALSE);
         c.exec(sql);
         if( !c.fetchNext() )
            continue;
         if( n == 0 )
            continue;
         if( n > 100 )
            message(">100 SICs targeted");
      }

      Cursor c;
      CString sic;
      c.bind(sic);
```

```
    char sql[512] = "select siccode from placement_sice where ad_id=";
    addValue(sql, ad.id, FALSE);
    c.exec(sql);
    SICCode *s = 0;
    while( c.fetchNext() ) {
       stripSpaces(sic);
       if( s == 0 ) {
          // to do: count the # of SICs first, and allocate that number
          //        rather than 50
          s = new SICCode[n];
          ad.sicCodes = s;
       }
       *s = sic;
       if( ++ad.nSICCodes == n ) {
          ASSERT( !c.fetchNext() );
          break;
       }
       s++;
    }
 }

 // load regional
 for( i = 0; i < ads.GetSize(); i++ ) {
    Region *l = 0;
    Ad &ad = *ads.GetAt(i);
    if( !ad.isTargeted() )
       continue;

    int n = 0;
    {
       Cursor c;
       c.bind(SQL_C_LONG, &n, sizeof(n));
       char sql[512] = "select count(*) from placement_locations where ad_id=";
       addValue(sql, ad.id, FALSE);
       c.exec(sql);
       if( !c.fetchNext() )
          continue;
       if( n == 0 )
          continue;
       if( n > 100 )
          message("->100 locations targeted");
    }

    Cursor c;
    WORD country;
    CString state, zip;
    int areaCode;
    c.bind(SQL_C_LONG, &country, sizeof(country));
    c.bind(state);
    c.bind(zip);
    c.bind(SQL_C_LONG, &areaCode, sizeof(areaCode));
    char sql[512] = "select country,state,zipcode,areacode from placement_locations where ad=
    addValue(sql, ad.id, FALSE);
    c.exec(sql);
    areaCode = 0;
    while( c.fetchNext() ) {
       if( l == 0 ) {
          l = new Region[n];
          ad.locations = l;
       }
       l->country = country;
       l->state = state;
       l->zipCode = zip;
       l->areaCode = areaCode;
       if( ++ad.nLocations == n ) {
          ASSERT( !c.fetchNext() );
          break;
       }
       l++;
       areaCode = 0;
    }
 }

 latmain.commit();
```



```
    if( ads.GetSize() == 0 && forTargeting ) {
       // db connection down, use some default ads
       makeDefaultAds(ads);
    }

    if( defaultAd == 0 ) {
       TPACE("no default ad\n");
       message("no default ad");
    }

    return ads.GetSize() != 0 && defaultAd != 0;
 }
```

```
// sqldb.cpp
//

#include "stdafx.h"
#include <fstream.h>
#include "objects.h"
#include "/d/toolkit/db.h"
#include "/d/toolkit/iaf_util.h"
#include "/d/toolkit/dbutil.h"
#include "/d/toolkit/dbpool.h"
#include "/d/toolkit/crit.h"

// this ad displayed if a bad sitekey is encountered
const cBadKeyAdID = 49;

extern CriticalSection fast;

Database iafmain;
void message(const char *);

BOOL defaultAdsMode = FALSE;

static int uctOfs;
static void localToUCT(time_t& t)
{
   t += uctOfs;
}

// This is temporary.  Used for non-unique users.
// Eventually will be smarter about what to send to
// these users.
Ad *defaultAd = 0;

Ad *badKeyErrorAd = 0;

typedef CArray<Ad *, Ad *> AdArray;

BOOL loadAds(AdArray& ads,
         DWORD advertiserID,     // 0=all
         BOOL forTargeting,      // if forTargeting, update Ad::targetSites to reflect
                                 //   site exclusions
         BOOL activeOnly,        // active=1 only
         BOOL includeExpired,    // include where enddate has past or where all delivered
         BOOL newestFirst,       // order from newest to oldest
         DWORD siteID = 0);

BOOL openSQLDB()
{
   iafmain.open();
   openDBPool();

// if( !iafmain.open() )
//    return FALSE;
//
// if( !openDBPool() )
//    return FALSE;

// if( !iafmain.Open(0, FALSE, FALSE,
//                   "ODBC;DSN=iaf;UID=sa;PWD=",
//                   /*FALSE*/ TRUE) )
//    return FALSE;

   if( !loadAds(ads, 0, TRUE, TRUE, FALSE, FALSE) )
      return FALSE;

   return TRUE;
}

void reloadAds()
{
   BOOL ok = FALSE;
   message("waiting to reload ads...");
   AfxGetApp()->m_pMainWnd->Invalidate();
   AfxGetApp()->m_pMainWnd->UpdateWindow();
   while( 1 ) {
```



```
      {
         Crit c(fast);
         if( allFree() ) {
            for( int i = 0; i < ads.GetSize(); i++ ) {
               delete ads.GetAt(i);
               ads.RemoveAll();
               defaultAd = 0;
               ok = loadAds(ads, 0, TRUE, TRUE, FALSE, FALSE);
            }
            break;
         }
      }
      Sleep(50);
   }
   if( ok )
      message("Ad reload completed OK");
   else
      message("Ad reload failure!");
}

// note: this isn't getting called yet
void closeSQLDB()
{
   iafmain.close();
}

//---------------------------------------------------------------------
// Ads

AdArray ads;

class AdCursor : public Cursor
{
public:
   AdCursor()
   {
      bind(SQL_C_LONG, &ad.id, 4);
      bind(SQL_C_LONG, &ad.type, sizeof(ad.type));
      bind(SQL_C_LONG, &ad.os, sizeof(ad.os));
      bind(SQL_C_LONG, &ad.browser, sizeof(ad.browser));
      bind(SQL_C_LONG, &ad.domainType, sizeof(ad.domainType));
      bind(SQL_C_LONG, &ad.isp, sizeof(ad.isp));
      bind(ad.fileName);
      bind(ad.jumpTo);
      bind(SQL_C_LONG, &ad.frequency, sizeof(ad.frequency));
      bind(SQL_C_LONG, &ad.imageSeries, sizeof(ad.imageSeries));
      bind(SQL_C_LONG, &ad.maxImpressions, sizeof(ad.maxImpressions));
      bind(SQL_C_LONG, &ad.nShown, sizeof(ad.nShown));
      bind(SQL_C_LONG, &ad.startTime, sizeof(ad.nShown));
      bind(SQL_C_LONG, &ad.endTime, sizeof(ad.nShown));
      bind(SQL_C_LONG, &ad.flags, sizeof(ad.flags));
      bind(SQL_C_LONG, &ad.hoursOfDay, sizeof(ad.hoursOfDay));
      bind(SQL_C_LONG, &ad.daysOfWeek, sizeof(ad.daysOfWeek));
      bind(SQL_C_LONG, &ad.nEmployees, sizeof(ad.nEmployees));
      bind(SQL_C_LONG, &ad.salesVolume, sizeof(ad.salesVolume));
      bind(SQL_C_LONG, &ad.active, sizeof(ad.active));
      bind(ad.adDescription);
      bind(SQL_C_LONG, &ad.maxAmount, sizeof(ad.maxAmount));
      bind(ad.szPONumber);
      bind(SQL_C_LONG, &ad.approved, sizeof(ad.approved));
      bind(SQL_C_LONG, &ad.nJumps, sizeof(ad.nJumps));
   }

   Ad ad;
};

// ... TODO!!! This function is not thread-safe.
void recalcSI()
{
   for( int i = 0; i < ads.GetSize(); i++ ) {
      Ad& ad = *ads.GetAt(i);
      ad.calcSI();
   }
}
```

```
static void makeDefaultAds(AdArray& ads)
{
   ifstream defAds("c:\\ian\\default_ads.txt");
   if( !defAds.is_open() ) {
      ASSERT(FALSE);
      return;
   }

   message("db connection failed, using default_ads.txt");
   defaultAdsMode = TRUE;

   while( 1 ) {
      char fn[128];
      char jumpTo[128];
      *fn = 0;
      defAds >> fn >> jumpTo;
      if( *fn == 0 )
         break;

      Ad& ad = *(new Ad);
      defaultAd = &ad;
      time_t now;
      ad.startTime = time(&now) - 60 * 60 * 24 * 15;
      ad.endTime = now + 60 * 60 * 24 * 15;
      ad.fileName = fn;
      ad.jumpTo = jumpTo;
      ads.Add(&ad);
   }
}

BOOL loadAds(AdArray& ads,
         DWORD advertiserID,     // 0=all
         BOOL forTargeting,      // if forTargeting, update Ad::targetSites to reflect
                                 //   site exclusions
         BOOL activeOnly,        // active=1 only
         BOOL includeExpired,    // include where enddate has past or where all delivered
                                 //   (for management and reporting...)
         BOOL newestFirst,       // order from newest to oldest
         DWORD approveSiteID)    // exclude ads the specified site has approved

{
   {
      // calc time zone adjustment
      CTime t = CTime::GetCurrentTime();
      tm gmt, local;
      t.GetGmtTm(&gmt);
      t.GetLocalTm(&local);
      if( local.tm_hour > gmt.tm_hour )
         gmt.tm_hour += 24;
      uctOfs = (gmt.tm_hour - local.tm_hour) * 60 * 60;
   }

   ads.SetSize(0, 64);

   DWORD active = 1;
   getConfigValue("Active", active);
   AdCursor rs;
   char sql[2000] =
      "select id,type,os,browser,domainType,isp,filename,jumpto,frequency,image_series,\
max_impressions,n_shown,datediff(ss, '1/1/70', start_time),datediff(ss, '1/1/70', end_time),\
flags,hours_of_day,days_of_week,employees,sales,active,description,max_amount,po_number,\
approved,n_jumps from placements";

   BOOL where = FALSE;

   if( !includeExpired ) {
      strcat( sql, " where (max_impressions=0 or n_shown<max_impressions) and \
(end_time=null or end_time>=getdate())");
      where = TRUE;
   }

   if (activeOnly)
   {
      if( where ) {
         strcat(sql, " and");
      } else
```



```
         where = TRUE;
         strcat(sql, " where");
      }
      strcat( sql, " active=" );
      addValue(sql, active, FALSE);
   }

   if( advertiserID ) {
      if( where ) {
         strcat(sql, " and");
      } else {
         where = TRUE;
         strcat(sql, " where");
      }
      strcat(sql, " advertiser=");
      addValue(sql, advertiserID, FALSE);
   }

   if( approveSiteID ) {
      if( where ) {
         strcat(sql, " and");
      } else {
         where = TRUE;
         strcat(sql, " where");
      }

      strcat(sql, " not exists (select * from approved where site_id=");
      addValue(sql, approveSiteID, FALSE);
      strcat(sql, " and ad_id=id)");
   }

   if( newestFirst ) {
      strcat(sql, " order by id desc");
   }

   rs.exec(sql);

   while( 1 ) {
      // defaults in case null
      rs.ad.flags = 0;

      if( rs.fetchNext() )
         break;

      // if for debug, don't load.  You can make this test a registry
      // setting if you like so that you can load debug records, or
      // add a cmd line setting.
      if( !rs.ad.isProduction() )
         continue;

//    if( rs.isNull(12) ) {
//       time_t now;
//       rs.ad.startTime = time(&now);
//       rs.ad.endTime = rs.ad.startTime + 60 * 60 * 24 * 30;
//    }
//    else {
//       localToUCT(rs.ad.startTime);
//       localToUCT(rs.ad.endTime);
//    }
      if( rs.isNull(12) ) {
         // ad server needs fake times for now...
         if( forTargeting ) {
            time_t now;
            rs.ad.startTime = time(&now) - 60 * 60 * 24 * 15;
            rs.ad.endTime = now + 60 * 60 * 24 * 15;
         }
         else {
            rs.ad.startTime = rs.ad.endTime = 0;
         }
      }
      else {
         localToUCT(rs.ad.startTime);
         localToUCT(rs.ad.endTime);
      }
```

```
void Request::service()
{
   const char *p = strchr(request, ' ');
   if( p )
      fileName = CString(request, p - request);
   else
      fileName = request;

   {
      const char *p = fileName;
      if( *p == '/' )
         p++;
      if( *p == 0 ) {
         // send default
         //sendFile("k:\\my documents\\internet address finder\\iafmain.htm");
#if defined(_IAF)
         sendFile("c:\\iaf\\html\\iafmain.htm");
         return;
#endif
      } else {
         if( strchr(p, '\\') == 0 && strstr(p, "..") == 0 ) {
            if( strchr(p, '/') != 0 ) {
               CString f = "c:\\ian\\";
               f += p;
               sendFile(f);
               return;
            }
            else
            {
#if defined(_IAF)
               CString f = "c:\\iaf\\html\\";
#elif defined(_MANAGE)
               CString f = "c:\\ian\\manage\\";
#else
               ASSERT(FALSE);
               CString f = "jekldf";
               //CString f = "k:\\my documents\\ad federation\\";
#endif
               f += p;
               sendFile(f);
               return;
            }
         }
      }
      sendError(c, "404 Not Found");
   }
}

void Request::sendInternalError()
{
   sendError(c, "500 Internal Server Error");
}
```

```
// rememberad.cpp
//

#include "stdafx.h"
#include "objects.h"
#include "rememberad.h"
#include "/d/toolkit/hashwf.h"
#include "/d/toolkit/crit.h"

const SZ = 107311;

// this is a test
static int cr;
#define INCRIT { ASSERT(cr++==0); cr++; }
#define OUTCRIT { ASSERT(cr--==1); cr--; }

void message(const char *);

extern CriticalSection fast;

struct Key
{
   DWORD userID;
   DWORD fromHash;

   BOOL operator==(const Key& k) const
   {
      return userID == k.userID && fromHash == k.fromHash;
   }

   void setID(User *u)
   {
      if( u->userID )
         userID = u->userID;
      else
         userID = u->ip;
   }

   void setFrom(const char *from)
   {
      fromHash = hashwf(from);
   }
};

UINT HashKey(Key key)
{
   return key.userID ^ key.fromHash;
   // default identity hash - works for most primitive values
// return ((UINT)(void*)(DWORD)key) >> 4;
}

struct Value
{
   DWORD adSent;
   DWORD time;
};

class Memory
{
public:
   Memory() : sent(100)
   {
      sent.InitHashTable(SZ);
   }

   void remember(Key& k, DWORD adID);
   DWORD lookup(Key& k);

private:
   void purge();

   CMap<Key, Key&, Value, Value&> sent;
} memory;

// todo: fix
```



```
// todo: nonunique hashes
//
//DWORD hash(const char *from, User *u)
//{
// char buf[10];
// wsprintf( buf, "%lx", u->getID() );
// CString s = buf;
// s += from;
// return hashwf(s);
//}

void Memory::remember(Key& k, DWORD adID)
{
   static int count;
   if( ++count > 1000 ) {
      count = 0;
      purge();
   }

   Value v;
   v.adSent = adID;
   v.time = ::GetTickCount();
   sent.SetAt(k, v);
}

DWORD Memory::lookup(Key& k)
{
   Value value;
   if( sent.Lookup(k, value) ) {
      return value.adSent;
   }
   return 0;
}

void Memory::purge()
{
   const LIMIT = 1000 * 60 * 60 * 24;      // too much?

   if( sent.GetCount() > SZ ) {
      message("remember map > SZ");
   }

   DWORD now = ::GetTickCount();
   POSITION p = sent.GetStartPosition();
   while( p ) {
      Key k;
      Value v;
      sent.GetNextAssoc(p, k, v);
      if( now - v.time > LIMIT )
         sent.RemoveKey(k);
   }
}

void rememberSend(Ad *ad, User *u, const char *fromDoc)
{
   Crit c(fast);
// INCRIT
   Key k;
   k.setID(u);
   k.setFrom(fromDoc);
   memory.remember(k, ad->id);
// OUTCRIT
}

DWORD queryAdSent(User *u, const char *fromDoc)
{
   Crit c(fast);
// INCRIT
   Key k;
   k.setID(u);
   k.setFrom(fromDoc);
   DWORD d = memory.lookup(k);
// OUTCRIT
   return d;
}
```

```
// a truly random distribution is used for them rather than
// leftovers.
static int testCounter;
if( ++testCounter % 4 == 0 ) {      // just try every 4 to save CPU
   // test ad avail?
   lowestSI = 1051;
   int i = start;
   while( 1 ) {
      Ad& ad = *ads.GetAt(i);
      if( ad.type == Test && ad.si < lowestSI && ad.criteriaOK(db, user, page) )
      {
         lowestSI = ad.si;
         adLowestSI = &ad;
      }
      i = (i + 1) % nAds();
      if( i == start )
         break;
   }
   if( lowestSI <= 1050 )
      return adLowestSI;
}

lowestSI = SIMAX;
adLowestSI = defaultAd;

// Check remnants first.  This way, we don't
// have to do ad matching for any targeted ads
// with high SI's.
//
int i = start;
while( 1 ) {
   Ad& ad = *ads.GetAt(i);
   if( ad.type == Normal && !ad.isTargeted() && ad.si < lowestSI && ad.spreadOK(page) )
   {
      lowestSI = ad.si;
      adLowestSI = &ad;
   }

   i = (i + 1) % nAds();
   if( i == start )
      break;
}

// this is temp; eventual all placements will have book rates
// you'll want to remove this to get better performance (no ad matching
// if remnant has worst SI).
static int counter;
if( ++counter & 1 ) {
   // for ads with no booking amount,
   // allow a targeted ad to run sometimes
   if( lowestSI == 1100 )
      lowestSI--;
}

   // for ads where we don't care about # impressions,
   // bias in favor of targeted
//    if( lowestSI == 1100 )
//       lowestSI--;

// todo later: if ads are sorted by si (lowest first),
//             you can quit matching as soon as you find
//             one.  Could be a good optimization.

// do targeted
i = start;
while( 1 ) {
   Ad& ad = *ads.GetAt(i);
   if( ad.type == Normal && ad.isTargeted() &&
      ad.si < lowestSI &&
      ad.spreadOK(page) &&
      ad.matches(user, page) &&
      ad.exposuresOK(db, user) )
   {
      // found a good one
      lowestSI = ad.si;
```



```
      adLowestSI = &ad;
   }

   i = (i + 1) % nAds();
   if( i == start )
      break;
}

if( lowestSI > 1400 ) {
   // do either a barter ad or an fan dev ad
   static int counter;
   if( ++counter % 5 == 0 ) {
      // do an fan dev ad
      i = start;
      while( 1 ) {
         Ad& ad = *ads.GetAt(i);
         if( ad.type == FanDev && ad.criteriaOK(db, user, page) ) {
            // found a good one
            adLowestSI = &ad;
            break;
         }
         i = (i + 1) % nAds();
         if( i == start )
            break;
      }
   }
   else {
      // do barter
      lowestSI = SIMAX;
      i = start;
      while( 1 ) {
         Ad& ad = *ads.GetAt(i);
         if( ad.type == Barter &&
            ad.si < lowestSI &&
            ad.criteriaOK(db, user, page) ) {
            // found a good one
            adLowestSI = &ad;
            lowestSI = ad.si;
         }
         i = (i + 1) % nAds();
         if( i == start )
            break;
      }
   }
}

return adLowestSI;
}
```

```
// request.cpp
//

#include "stdafx.h"
#include "/d/toolkit/sock.h"
#include "request.h"
#include "/d/toolkit/iaf_util.h"

#if defined(_CONSOLE)
#include <fstream.h>
#endif

#if defined(_IAF)
extern ofstream *outLog;
void impression();
#endif

extern CString gratuitous;

Request::Request(
   Connection *_c,
   Verb _v,
   const char *_request,
   const sockaddr_in &from) :
      c(_c), request(_request), v(_v)
{
   userIP = from.sin_addr.s_addr;
}

int spider = 0;

BOOL Request::sendFile(const char *fileName, const char *insertStr)
{
#if defined(_IAF)
   *outLog << "-send " << fileName << " " << inet_ntoa( (in_addr&) userIP ) << '\n';
#endif

   const char insertChar = '~';
   BOOL isSpider = FALSE;

   CString hdr = "HTTP/1.0 200 OK\r\nContent-Type: ";
   if( strstr(fileName, ".class") != 0 ) {
      hdr += "application/java\r\nContent-Length: ";
   }
   else if( strstr(fileName, ".gif") != 0 ) {
      hdr += "image/gif\r\nContent-Length: ";
   } else {
      hdr += "text/html\r\nContent-Length: ";
#if defined(_IAF)
      impression();
#endif
      int gnt = 0;
      if( strstr(request, "-Agent: Lycos") != 0 )
         gnt = 1;
       if( strstr(request, "InfoSeek Robot") != 0 )
         gnt = 2;
      if( strstr(request, "-Agent: WebCrawl") != 0 )
         gnt = 3;

      if( gnt )
      {
         isSpider = TRUE;
         spider++;
         #if defined(_CONSOLE)
         cout << "********** Robot " << gnt << " **********\n";
         #endif
      }
   }

   const BUFSIZE = 130000;
   char buf[BUFSIZE + 260];
   CFile f;
   CFileException fe;
```



```
   if( v == GET || v == POST ) {
      if( !f.Open(fileName, CFile::modeRead | CFile::shareDenyWrite, &fe) ) {
         if( fe.m_cause == CFileException::accessDenied )
            sendError(c, "404 Not Found (Access Denied)");
         else if( fe.m_cause == CFileException::sharingViolation )
            sendError(c, "404 Not Found (Sharing Violation)");
         else
            sendError(c, "404 Not Found");
         return FALSE;
      }
      n = f.Read(buf, BUFSIZE);
   }
   else {
      isSpider = FALSE;

      // HEAD
      n = getFileSize(fileName);
      if( n == 0 ) {
         sendError(c, "404 Not Found");
         return FALSE;
      }
   }
   ASSERT( n != 0 && n != BUFSIZE );

   char *p = buf;
   if( insertStr ) {
      while( 1 ) {
         p = strchr(p, insertChar);
         if( p == 0 )
            break;
         int l = strlen(insertStr);
         memmove(p + l, p + 1, strlen(p+1));
         memcpy(p, insertStr, l);
         p += l;
         n += l;
      }
   }

   if( isSpider ) {
      if( gratuitous.IsEmpty() ) {
      #if defined(_CONSOLE)
         cout << "gratuitous empty. (?)\n";
      #endif
      }
      else {
         buf[n] = 0;
         char *p = strstr(buf, "</BODY");
         if( p ) {
            for( int i = 0; i < 20; i++ ) {
               strcpy(p, gratuitous);
               p += gratuitous.GetLength();
            }
            strcpy(p, "</BODY></HTML>");
            n = (p - buf) + 14;
         }
         else {
         #if defined(_CONSOLE)
                  cout << "/body?\n";
         #endif
         }
      }
   }

   char temp[100];
   itoa(n, temp, 10);   // content length
   hdr += temp;
   hdr += "\r\n\r\n";

   c->write( (const char *) hdr, hdr.GetLength() );
   if( v == GET || v == POST )
      c->write(buf, n);

   return TRUE;
}
```

```
// match.cpp
//
// Ad Matching!
//

#include "stdafx.h"
#include "objects.h"
#include "/d/toolkit/db.h"
#include "/d/toolkit/dbutil.h"

extern Ad *defaultAd;
extern Ad *badKeyErrorAd;

extern int nextAd;

int nAds();

// Returns TRUE if this location is in region.
//
BOOL Location::in(const Region& region)
{
   if( region.country != 0 && country != region.country )
      return FALSE;

   if( region.areaCode != 0 && areaCode != region.areaCode )
      return FALSE;

   if( !region.state.IsEmpty() && stricmp(state, region.state) != 0 )
      return FALSE;

   if( region.zipCode.IsEmpty() )
      return TRUE;

   // zip
   CString myZip = zipCode.Left(5); // strip zip+4 for now
   CString regZip = region.zipCode.Left(5);
   CString regZipEnd = region.zipEnd.Left(5);

   if( regZipEnd.IsEmpty() )
      return regZip == myZip;

   return myZip >= regZip && myZip <= regZipEnd;
}

BOOL Ad::exposuresOK(Database& db, User *user)
{
   seriesNext = 0;

   if( frequency == 0 || &db == 0 )
      return TRUE;

   int n;
   BOOL found;
   {
      if( user->getID() == 0 ) {
         TRACE("userId=0?\n");
         return FALSE;
      }

      Cursor c(db);
      c.bind( SQL_C_LONG, &n, sizeof(n) );
      char sql[512] = "select exposures from exposures where ad_id=";
      addValue(sql, id, FALSE);
      strcat(sql, " and user_id=");
      addValue(sql, user->getID(), FALSE);
      c.exec(sql);
      found = c.fetchNext();
   }
   if( found ) {
      if( n >= frequency )
         return FALSE;

      seriesNext = n + 1;

      char sql[1024] =
```



```
         "update exposures set exposures=exposures+1 where ad_id=";
      addValue(sql, id, FALSE);
      strcat(sql, " and user_id=");
      addValue(sql, user->getID(), FALSE);
      db.exec(sql);

      return TRUE;
   }

   char sql[1024] =
      "insert exposures values(";
   addValue(sql, id);
   addValue(sql, user->getID(), FALSE);
   strcat(sql, ",1)");
   db.exec(sql);

   return TRUE;
}

// Note: any matching required for nontargeted ads can be placed here,
//       since this function is called for both targeting and untargeted
//       ads.
//
BOOL Ad::spreadOK(SitePage *sitepage)
{
   // is start time met?
   if( !started ) {
      time_t now;
      if( time(&now) < startTime )
         return FALSE;
      started = TRUE;
   }

   // impressions OK?
   if( nShown >= maxImpressions && maxImpressions != 0 )
      return FALSE;

   if( isSpreadEvenly() && ei >= 1120 )
      return FALSE;

   if( !targetSites.IsEmpty() ) {
      if( sitepage == 0 )
         return FALSE;
      BOOL v;
      BOOL found = targetSites.Lookup(sitepage->siteID, v);
      if( includeSites ) {
         // if we have pages to target too, ok if site
         // doesn't match (check if page does next).
         if( !found && targetPages.IsEmpty() )
            return FALSE;
      } else if( found )
         return FALSE;
   }

   return TRUE;
}

// Does user and site match this ad's criteria?
BOOL Ad::matches(User *user, SitePage *sitepage)
{
   if( !targetPages.IsEmpty() ) {
      if( sitepage == 0 )
         return FALSE;
      BOOL v;
      BOOL found = targetPages.Lookup(sitepage->id, v);
      if( includePages ) {
         if( !found )
            return FALSE;
      }
      else if( found )
         return FALSE;      // excluding this page
   }

   // Operating system
   DWORD o = 1 << ((int) user->os);
```

```
    if( (o & os) == 0 )
        return FALSE;

    // Browser
    o = 1 << ((int) user->browser);
    if( (o & browser) == 0 )
        return FALSE;

    // DomainType
    int userISP = 0;
    int dt = (int) user->domainType;
    if( dt >= (int) dtISPOther ) {
        userISP = dt - (int) dtISPOther + 1;
        dt = 0;

        // ISP
        o = 1 << userISP;
        if( (o & isp) == 0 )
            return FALSE;
    }
    else {
        o = 1 << dt;
        if( (o & domainType) == 0 )
            return FALSE;
    }

    // location
    if( locations != 0 ) {
        if( userISP )           // if ISP, don't know location (yet)
            return FALSE;

        BOOL ok = FALSE;
        for( int i = 0; i < nLocations; i++ ) {
            if( user->location.in( locations[i] ) ) {
                ok = TRUE;
                break;
            }
        }
        if( !ok )
            return FALSE;
    }

    // hour of day / day of week
    if( hoursOfDay != 0xffffff || daysOfWeek != 0x7f ) {
        tm *t;
        if( isAbsoluteTime() ) {
            // EST time relative
            time_t now;
            time(&now);
            t = localtime(&now);
        }
        else {
            t = user->location.userRelativeTime();
            if( t == 0 )
                return FALSE;
        }
        if( (hoursOfDay & (1 << t->tm_hour)) == 0 )
            return FALSE;
        if( (daysOfWeek & (1 << t->tm_wday)) == 0 )
            return FALSE;
    }

    // sales
    if( salesVolume != 0x7fffffff ) {
        o = 1 << user->salesVolume;
        if( (o & salesVolume) == 0 )
            return FALSE;
    }

    // # employees
    if( nEmployees != 0x7fffffff ) {
        o = 1 << user->nEmployees;
        if( (o & nEmployees) == 0 )
            return FALSE;
    }
```



```
    // SIC
    if( nSICCodes ) {
        BOOL ok = FALSE;
        int i = 0;
        while( 1 ) {
            if( i >= nSICCodes ) {
                // no match
                return FALSE;
            }
            SICCode& pattern = sicCodes[i];
            user->sicCodes.reset();
            SICCode sc;
            while( user->sicCodes.getNext(sc) ) {
                if( pattern.matches(sc) ) {
                    ok = TRUE;
                    break;
                }
            }
            if( ok )
                break;
            i++;
        }
    }

    // Site and page categories
    // Do last, because this is expensive (disk hit)
    if( !siteCategories.isEmpty() ) {
        BOOL v;
        if( sitepage == 0 )
            return FALSE;
        sitepage->loadCategories();
        for( int i = 0; i < sitepage->categories.GetSize(); i++ )
            if( siteCategories.Lookup(sitepage->categories.GetAt(i), v) )
                return TRUE;
        return FALSE;
    }

    return TRUE;
}

inline BOOL Ad::criteriaOK(Database& db, User *user, SitePage *page)
{
    return spreadOK(page) &&
        (!isTargeted() ||
         (matches(user, page) && exposuresOK(db, user))
        );
}

// todo: if reload ads, need to handle the fact that
//       one may still be in use and can't just delete.
//       (Crit sect released during sending of file.)
//
Ad* Ad::getAd(Database& db, User *user, SitePage *page, BOOL increment)
{
    const SIMAX = 1000000;

    if( user->uniqueness < uLikely )
        return defaultAd;

    if( page == 0 ) {
        if( badKeyErrorAd )
            return badKeyErrorAd;
        ASSERT(FALSE);
    }

    if( increment )
        nextAd = (nextAd + 1) % nAds();

    int lowestSI;
    Ad *adLowestSI;

    const int start = nextAd;

    // Do a test ad, if appropriate.  Always do these first so that
```

```
    string.flush();
}
    // temp. just return first ad (155)
    //return new Ad( ads.ElementAt(0) );
    return new Ad( *defaultAd );
// return 0;
}
}
}
```

```
// cookie.cpp
//

#include "stdafx.h"
#include "objects.h"

//--------------------------------------------------------------------
// Cookie

const Cookie& Cookie::operator=(const char *s)
{
   sscanf(s, "%lx", &value);
   return *this;
}

/*static*/
Cookie Cookie::alloc(DWORD userID)
{
   ASSERT( userID != 0 );
   Cookie k;
   k.value = userID;
   return k;
}

// Get value for a particular cookie name from the HTTP header
//   hdr - points to the Cookie: field in the header
//
void Cookie::getFromHeader(const char *hdr, const char *name)
{
   hdr += 7;   // skip "Cookie:"

   const char *p = strchr(hdr, '\r');
   if( p ) {
      CString nm = name;
      nm += '=';
      const char *q = strstr(hdr, nm);
      if( q && q < p )
         *this = q + nm.GetLength();
   }
}
```

```
    text << "<b>location:    </b>";
    if( location.country == 256 ) {
       text << "USA";
    } else {
       text << "country #" << location.country;
    }
    text << "\r\n";

    text << "<b>job function:</b>" << "\r\n";
    text << "<b>gender:      </b>";
    if( gender == 'm' || gender == 'M' )
       text << "Male";
    else if( gender == 'f' || gender == 'F' )
       text << "Female";
    else
       text << '?';
    text << "\r\n";
    text << "<hr>";

    Domain *d = Domain::lookupDomain(ip);
    if( d == 0 ) {
       text << "<i>No company information available.</i>";
    }
    else {
       text << "<b>domain name: </b>" << (const char *) d->domain << "\r\n";      //domainName
       text << "<b>bus. name:   </b>" << (const char *) d->name <<"\r\n";
       text << "<b>address:     </b>" << (const char *) d->address[0] << "\r\n";
       for( int i = 1; i < NADDR; i++ ) {
          if( !d->address[i].IsEmpty() ) {
             text << "                " << (const char *) d->address[i] << "\r\n";
          }
       }

       text << "<b>contact:     </b>" << (const char *) d->contact[0] << "\r\n";
       for( i = 1; i < NCONTACT; i++ ) {
          if( !d->contact[i].IsEmpty() ) {
             text << "                " << (const char *) d->contact[i] << "\r\n";
          }
       }

       text << "<b>industries:  </b>\r\n";
       {
          sicCodes.reset();
          SICCode sc;
          while( sicCodes.getNext(sc) ) {
             text << "                " << sc.asTextFullyPadded() << "\r\n";
          }
       }

//     for( i = 0; i < MAXSICS; i++ )
//        if( d->sicCodes[i] )
//           text << d->sicCodes[i] << ' ';

       text << "\r\n<b>no. empl.:   </b>";
       if( nEmployees )
          text << nEmployees;
       else
          text << "less than 25 (unknown)";
       text << "\r\n";
       text << "<b>revenue:     </b>";
       if( salesVolume )
          text << salesVolume;
       else
          text << "less than $1MM (unknown)";
       delete d;
    }

    text << "<hr>";
    text << "<b>You are interested in the following:</b>\r\n\r\n";
    text << "<b>Interest Level  Category  Description\r\n";
    text << "---------------  --------  -----------------------------------------\r\n";
    {
       DWORD level;
       CString category;
```



```
       CString desc;
       Cursor c(db);
       c.bind(SQL_C_LONG, &level, 4);
       c.bind(category);
       c.bind(desc);
       char sql[512];
       wsprintf(sql,
"select interest_level,category.name from interests,user_interests\
 where interests.id=interest_id and user_id=%ld\
 order by interest_level DESC", userID);
       c.exec(sql);
       while( c.fetchNext() ) {
          char buf[32];
          wsprintf(buf, "%14ld  ", level);
          text << buf;
          text << category << "  " << desc << "\r\n";
       }
       db.commit();
    }
}

void User::_getNetworkInfo(Database& db, BOOL *timedOut)
{
    if( ip == 0 ) {
       ASSERT(FALSE);
       return;
    }

// if( domainType != dtUnknown ) {
//    // got dt from header info
//    // if ISP/OSP, location and sales, etc. don't apply.
//    // if we have done a tracert, location does apply.
//    // for ISPs/OSPs,
//    if( domainType != dtNetcom )  // did tracert for netcom
//       return;
// }

    // Note: do the following for all domain types to at least get country.

    NetworkNumber n;
    n = justNetworkNumber(ip);

    char buf[256] =
       "select domain_type,sales,num_employees,sic,country,state,zipcode,areacode from networks";
    Cursor c(db);
    if( domainType < dtAOL ) {
       c.bind(SQL_C_LONG, &domainType, sizeof(domainType));
       c.bind(SQL_C_LONG, &salesVolume, sizeof(salesVolume)); salesVolume = 0;
       c.bind(SQL_C_LONG, &nEmployees, sizeof(nEmployees)); nEmployees = 0;
       sicCodes.bind(c);
    } else {
       strcpy(buf, "select country,state,zipcode,areacode from networks where netnumber=");
    }
    strcat( buf, n.sqlStr() );

    c.bind(SQL_C_LONG, &location.country, sizeof(location.country));
    c.bind(location.state);
    c.bind(location.zipCode);
    c.bind(SQL_C_LONG, &location.areaCode, sizeof(location.areaCode));
    if( timedOut != 0 )
       c.setTimeOut(1);
    c.exec(buf);
    if( c.timedOut() )
       *timedOut = TRUE;
    else
       c.fetchNext();

    if( uniqueness == uUnknown && (int) domainType >= (int) dtAOL )
       uniqueness = uUnlikely;

    if( domainType >= dtAOL ) {
       salesVolume = 0;
       nEmployees = 0;
       sicCodes.makeNull();
       if( domainType != dtNetcom && domainType != dtISPOther ) {
```

```
        // don't know location, except country
        location.state.Empty();
        location.zipCode.Empty();
        location.areaCode = 0;
      }
   }
   else {
      elcCodes.checkNull();
   }
}

#if !defined(_DERIVE)

const char cCookie[] = "Cookie:";

void User::liftVer(const char *verStr)
{
   int v1 = 0, v2 = 0;
   sscanf(verStr,
          "%d.%d", &v1, &v2);
   bVer1 = v1;
   bVer2 = v2;
}

/*
User* User::_lookupUserByID(DWORD userID)
{
   User *u = new User;
   return u;
}

User* User::_lookupUserByAddress(DWORD ip)
{
   DWORD userID = networkNodeTable.getUserID(ip, FALSE);
   if( userID == 0 ) {
      // Try to get domain info at least.  Note: if user is uniquely
      // identifiable, derive data process will create a record for the
      // user as soon as it gets a chance.
      userID = networkNodeTable.getUserID(justNetworkNumber(ip), TRUE);
   }
   if( userID ) {
      return _lookupUserByID(userID);
   }
   return 0;
}
*/

extern defaultAdsMode;

User* User::lookupUser(Database& db, DWORD ip, const char *requestHdr, BOOL loadDemographics, B

{
   BOOL _timedOut = &db == 0;
   BOOL *tmout = realTime ? &_timedOut : 0;

   //-------------------------------------------------
   // get cookie for lookup

   Cookie cookie;

   const char *ck = strstr(requestHdr, cCookie);
   if( ck )
      cookie.getFromHeader(ck, "IAF");

   //-------------------------------------------------
   // lookup

   User *u = 0;

   if( !cookie.isNull() ) {
      if( _timedOut ) {
         u = new User;
         u->uniqueness = uYes;
         u->ip = ip;
         u->userID = cookie.value;
         u->timedOut = TRUE;
```



```
      }
      else {
         // lookup by cookie
         u = _lookupUserByID(db, cookie.value, tmout);
         if( u ) {
            u->uniqueness = uYes;
            u->ip = ip;
         }
         else {
            if( defaultAdsMode ) {
               // db conn down
               u = new User;
               u->uniqueness = uYes;
               u->ip = ip;
               u->userID = cookie.value;
            } else {
               // Couldn't find user record, we will need to
               // assign a new cookie.  Do not load by IP, because
               // we don't want this user sharing a record
               // with others without cookies.
               //
               // Note: generally, this shouldn't happen.
               cookie.value = 0;
            }
         }
      }
   }
   else if( !_timedOut ) {
      u = _lookupUserByAddress(db, ip, tmout);
      if( u ) {
         u->ip = ip;
         u->hasCookie = FALSE;
      }
   }

   if( u == 0 ) {
      // make a default user object
      u = new User;
      //u->uniqueness = uNo;
      u->ip = ip;
      u->timedOut = _timedOut;
   }

   u->headerDerive(requestHdr);

   if( !cookie.isNull() )
      u->hasCookie = TRUE;

   if( loadDemographics && !_timedOut )
      u->_getNetworkInfo(db, realTime ? &u->timedOut : 0);

   return u;
}

//-------------------------------------------------------------------------
// SitePage

Ad* Ad::findSentTo(User *user, const char *fromDoc)
{
   DWORD adNum = querySentTo(user, fromDoc);

   for( int i = 0; i < nAds(); i++ ) {
      Ads ad = *ads.GetAt(i);
      if( ad.id == adNum )
         return new Ad(ad);
   }

   if( badKeyErrorAd && adNum == badKeyErrorAd->id )
      return badKeyErrorAd;

   if( user->uniqueness == uLikely ) {
#if defined(errLog)
      errLog << "findSentTo failed uniqueness=likely\n";
      errLog << "  user: " << user->userID << '\n';
      errLog << "  from doc: " << fromDoc << '\n';
```